# Circuit design with NAND and NOR

## Converting an expression containing AND, OR and INVERT into a circuit using NAND and NOR gates

by M. WARD, G3KZB*

AN ever increasing number of devices in the amateur's shack have circuits employing digital logic techniques: frequency counters, keyers, and—most recently—microprocessors. When the design of such a device is being considered, it is clearly an advantage to keep the variety of different ic packages to a minimum. A system using AND/OR gates will probably require inverters; but NAND/NOR has inversion built in. Further, with ics of the 7400 and cmos types, NAND/NOR packages are more common and cheaper than AND/OR/INVERT.

But *thinking* in terms of NAND/NOR is not easy. One way round the problem is to think and design in AND/OR; and then, as a last step, to convert into NAND/NOR. The procedure to be described is especially useful when each term and its complement can be obtained from flip-flops, in other words when there are available both A and $\overline{A}$, B and $\overline{B}$, and so on.

Suppose the expression to be realized with NAND/NOR gates is:

$$AB(C+\overline{D})+AE\overline{A}$$

(+ represents non-exclusive OR, ie A + B means A or B or both.) To make the explanation clearer, put a fullstop (".") for the Boolean AND operation:

$$A.B.(C+\overline{D})+A.E.\overline{A}$$

The aim is to break the expression down into separate terms: that is, each letter must stand on its own. First, simplify the Boolean expression. In this example, because $A.\overline{A}=1$, the last part of the expression reduces to E.

$$A.B.(C+\overline{D})+E$$

Next, use pairs of brackets to show the order in which the various operations of "." and "+" are to be carried out. This has been started already: the brackets indicate that $C+\overline{D}$ is to be worked out first. Normally one would *assume* that the operation "." takes precedence over "+", but for this purpose put in another pair of brackets:

$$(A.B.(C+\overline{D}))$$

And then the final step is +E. Writing the expression with enough space on its right hand side where later one can draw the circuit, this is labelled "Line 1".

Line 1: $(A.B.(C+\overline{D}))+E$

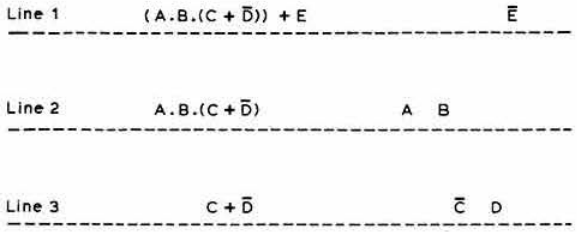
Leaving any single letters not within brackets (E in the example), carry the rest of the expression down to the next line, shedding the outermost pair(s) of brackets. This is Line 2.

Line 2: $A.B.(C+\overline{D})$

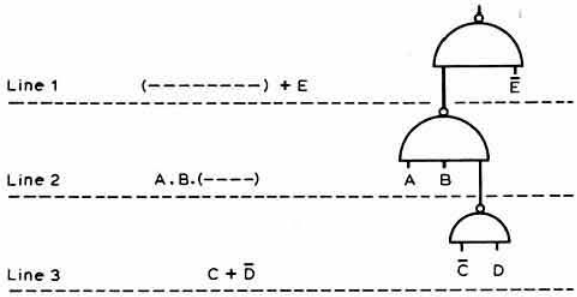
Repeat the process until all the brackets have gone.

Line 3: $C+\overline{D}$

Where the terms have separated out into single letters provides a guide for drawing the circuit. So, looking at the *even* lines, write the letters on the right-hand side, keeping their same relative positions. Then, for the odd lines write the *inverse* (the complement) of each term, ie A for $\overline{A}$, $\overline{A}$ for A, $\overline{B}$ for B, etc.



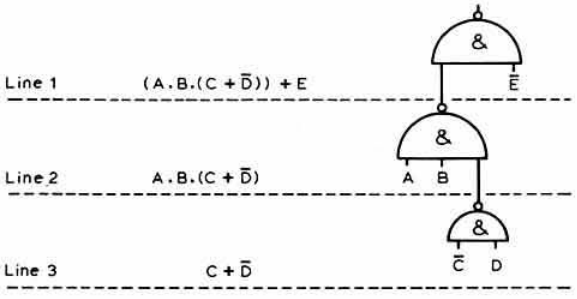

The letters will mark the position of inputs to gates. Other inputs will occur where there are brackets; ignoring for the moment the actual contents inside the outermost brackets on any line—suffice it that an input is drawn at this point.




Having the gates in outline, now label them NAND or NOR according to this schedule:

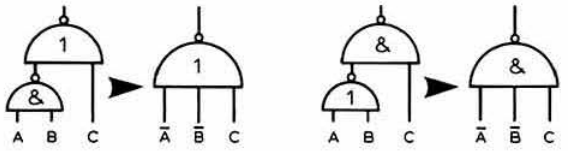
on odd lines "." gives rise to a NOR gate
on even lines "." gives rise to a NAND gate
on odd lines "+" gives rise to a NAND gate
on even lines "+" gives rise to a NOR gate.

On Line 1 of the example there is a "+" between the brackets and the E. From the schedule it is found that a NAND gate is called for. In Line 2 there are two "."; another NAND is indicated. And a final NAND on Line 3.




*Green Lane, Milford, Surrey.

An easy check is possible. If the expression is in its simplest form, all the gates will be of one kind only, because a NAND and a NOR in cascade can be simplified:




It is now possible to justify the rules given above. Gates have been drawn alongside each line. So any term which separates out into a single letter at Line 1 will pass through just one gate before it emerges at the top. A term on Line 2 will go through two gates, and so on. Passage through one NAND/NOR gate gives rise to an inversion. So, if the expression at the output contains the term A, $\overline{A}$ is required at the input on Line 1. Going through two NAND/NOR gates involves two inversions, restoring the original, since $\overline{\overline{A}} = A$. Hence, if the expression contains A, and A separates out on Line 2, write A at the input on that line.

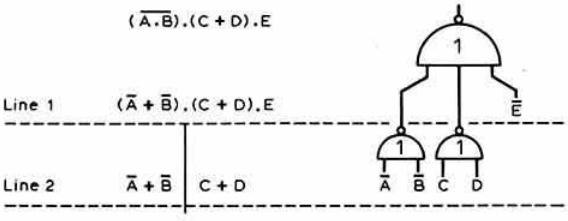
The expression A.B is obtained from a NOR gate with the two inputs $\overline{A}$ and $\overline{B}$. Similarly, A + B is obtained from a NAND gate with the same two inputs, $\overline{A}$ and $\overline{B}$. So the schedule works for Line 1.

Turning next to Line 2, one can work back from the expressions A.B and A + B through two gates. It will be seen that A.B derives from a NAND gate on Line 2; in like manner a NOR gate on this line produces an output of A + B. The discussion has related to Lines 1 and 2, but it could be generalized to show that the rules of procedure apply for all odd and all even lines.

If the expression contains any terms under a NOT bar that are not single letters, these need to be simplified first by applying de Morgan's Theorems:

$$\overline{A.B} = \overline{A} + \overline{B}$$
$$\overline{A+B} = \overline{A}.\overline{B}$$

This is illustrated in the next example:



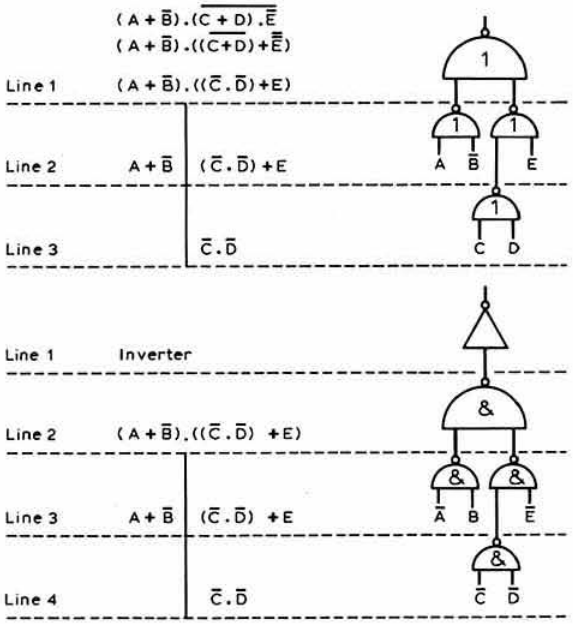

To emphasize the separation of Line 2 into two distinct parts (each of which gives rise to a gate), $\overline{A} + \overline{B}$ is kept apart from C + D with a vertical line.

Sometimes a rigorous algebraic simplification may not lead to the most efficient use of gates if:

(a) a particular expression is required in more than one place in the circuit;

(b) if a particular inverse is not already available so that an extra gate would have to be used as an inverter.

In these cases one may end up with a mixture of NANDs and NORs. Conversely, where the major part of a circuit consists of, say, NOR gates, in some sections the procedure described may work out to the NAND form. To obtain the NOR form when this happens, move all the lines down one and put an inverter on Line 1. For example:




## Summary

1. Simplify algebra.
2. Put in brackets, label Line 1.
3. Carry down all but single letters, shed outermost brackets, label Line 2.
4. Repeat as necessary.
5. Even lines: write letters for inputs.
6. Odd lines: write complements.
7. Draw outline gates.
8. Label gates:

| | | + |
|---|---|---|
| odd lines | NOR | NAND |
| even lines | NAND | NOR |

To change NAND ⟺ NOR: move down a line, put inverter on Line 1. □